ITL.1047US
(P17449)

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: **WRITE-BACK DISK CACHE**

INVENTOR: **JOHN I. GARNEY**

Express Mail No. EV 337934021 US
Date: December 3, 2003

Prepared by: Joseph Arrambide,, Trop, Pruner & Hu, P.C.
8554 Katy Freeway, Ste. 100, Houston, TX 77024
713/468-8880 ext 315 [Office], 713/468-8883 [Fax]

ITL.1047US
(P17449)

WRITE-BACK DISK CACHE

Background

This invention relates generally to disk caching for processor-based systems and more particularly to preserving coherency in a write-back disk cache.

Peripheral devices such as disk drives used in processor-based systems may be slower than other circuitry in those systems. There have been many attempts to increase the performance of disk drives. However, because disk drives are electromechanical in nature, there may be a finite limit beyond which performance cannot be increased.

One way to reduce the information bottleneck at the peripheral device, such as disk drives, is to use a cache. A cache is a memory device that logically resides between a device, such as a disk drive, and the remainder of the processor-based system. A cache is a memory location that serves as a temporary storage area for a device, such as the disk drive. Frequently accessed data resides in the cache after an initial access. Subsequent accesses to the same data may be made to the cache instead of to the disk drive.

Generally, two types of disk cache are used, write-through cache and write-back cache. Write-through disk cache means that the information is written both to the cache and to the corresponding disk drive. Write-back disk cache means that information is only written to the cache, and the information is only written to the corresponding disk drive when the data in the cache is being replaced with some other disk drive data. Write-back is faster than write-through cache (since writing to the slower disk is avoided on write operations) but may cause coherency problems since the data in the cache may be different (dirty) than in the corresponding disk drive. A cache line of data is dirty if the data in the cache line has been updated by the system but the corresponding disk drive has not been updated. A clean cache line is a line of data in the cache whose corresponding disk drive has the same data.

Caches are typically much smaller capacity compared to disk drives, but the most important data is kept in the cache for fastest access.

A processor-based system may use a write-back disk cache on a disk drive that is used during the normal operation of the computer and used to start (boot-up) the system. During the system start-up, the disk may be accessed by a basic input/output system (BIOS) disk request. Later in the start-up and after an operating system loads a disk drive software driver (operating system disk driver), the disk may be accessed by the operating system disk driver write request. However, the BIOS write request and the operating system disk driver write request may access or manipulate the cache and the disk drive inconsistently. Additionally, if a system crash occurs during an operation that precludes flushing the state of any dirty cache lines to the disk, the BIOS disk request may not access coherent data before the operating system disk driver loads.

Thus, there may be a need for alternative ways of implementing a disk cache.

Brief Description of the Drawings

Fig. 1 is a block diagram of a processor-based system in accordance with an embodiment of the present invention;

Fig. 2 is a flowchart in accordance with an embodiment of the present invention;

Fig. 3 is a flowchart in accordance with another embodiment of the present invention; and

Fig. 4 is a flowchart in accordance with another embodiment of the present invention.

Detailed Description

Referring to Fig. 1, the processor-based system 100 may be a desktop computer, a laptop computer, a server, or any of a variety of other processor-based systems.

The system 100 may include an input device 104 coupled to the processor 102. The input device 104 may include a keyboard or a mouse. The system 100 may also include an output device 106 coupled to the processor 102. The output device 106 may include a display device such as a cathode ray tube monitor, liquid crystal display, or a printer. Liquid crystal displays may use super twisted nematics (STN) or thin film transistor (TFT) technologies. Additionally, the processor 102 may be coupled to any number of memory devices such as a read only memory (ROM)

108, a random access memory (RAM) 110, an option ROM 112, a disk cache memory 114, or a disk drive 116. The disk drive 116 may be a floppy disk, hard disk, solid state disk, compact disk (CD) or digital video disk (DVD), or any other disk device that may be used in computer or consumer systems. In one embodiment, the system 100 may enable a wireless network access using a wireless interface 118. The wireless interface 118 may be a radio frequency interface, as one example, including a transceiver and an antenna. However, the present invention is not limited to processor-based systems that permit wireless access.

Disk cache 114 may be made from a ferroelectric polymer memory. Data may be stored in layers within the memory. The higher the number of layers, the higher the capacity of the memory. Each of the polymer layers includes polymer chains with dipole moments. Data may be stored by changing the polarization of the polymer between metal lines.

Ferroelectric polymer memories are non-volatile memories with sufficiently fast read and write speeds. For example, microsecond initial reads may be possible with write speeds comparable to those with flash memories.

In the typical operation of system 100, the processor 102 may access ROM 108 to execute a power on start-up test (POST) program and/or a basic input output system (BIOS) program. The processor may use the BIOS and POST software to initialize the system 100. The processor 102 may then access disk drive 116 to retrieve operating system software. The disk drive 116 may be a hard disk, floppy disk, or any other type of disk equivalent including solid state disk devices. The system 100 may also receive input from the input device 104 or may run an application program stored in memory or accessed from the wireless device 118. System 100 may also display the system 100 activity on the output device 106. The RAM 110 may be used to hold application programs or data that is used by processor 102. The disk cache 114 may be used to cache data for disk drive 116.

Referring to Fig. 2, an algorithm for disk caching in a processor-based system is disclosed. After the system start-up, block 200, which may also be referred to as a system boot or re-boot, the disk cache 114 (in Fig. 1) is probed to determine if the system was shutdown cleanly, as indicated in diamond 205. A clean shut-down in a system that has write-back cache is when the write-back cache has written back (flushed) any data that was dirty back to the disk drive. This results in there being no dirty data in the cache when the system is shutdown. If the system was shutdown cleanly, then write requests are monitored as shown in block 215. If the system was not shutdown cleanly, the dirty cache lines are written to the disk drive 116 (in Fig. 1), as indicated in block 210. A dirty cache line is a line of data that has been modified, but not flushed, and is therefore incoherent with the disk drive. After having cleaned the dirty cache lines, write requests are monitored as indicated in block 215. After a write request is detected and if the operating system disk driver has not been loaded, as illustrated in diamond 220, then the write request is logged, as indicated in block 225. After being logged, the write request is executed by writing to the disk drive, as indicated in block 230. Write requests are again monitored in block 215. However, if the operating system disk driver has been loaded, as indicated in diamond 220, then the cache lines are refreshed by reading disk locations which have data stored in the cache corresponding to disk location information previously logged by the option ROM as shown in block 235 and the algorithm is completed as indicated in block 240.

Referring to Figure 3, an algorithm 300 provides a BIOS/option ROM interface or communication protocol which may allow the software code stored in an option ROM 112, when executed, (Fig. 1) to monitor and execute write requests. During system start-up, the executed BIOS code may discover the presence of an option ROM 112 during the initialization of the system 100, as illustrated in block 310. In response to the discovery, the executed BIOS code may acknowledge to the option ROM 112 that the BIOS code supports the option ROM code filter function, which may include filtering disk requests, as illustrated in block 320. The option ROM filter function may include code for monitoring disk requests, such as interrupt 13 disk requests, and writing to disk drives.

When the executed BIOS code discovers a disk drive that may support disk caching, the BIOS code may invoke the option ROM entry point that communicates to the option ROM that the cache supported disk drive has been discovered so that BIOS can communicate the disk drive identification data to the option ROM 112, as illustrated in block 330. Then, the executed BIOS code may allow the option ROM 112 code to read and write to the disk drive 116 of Fig. 1, as in block 340. The option ROM 112, by executing code, may determine if the disk drive is cached; based on reading, for an example, a disk drive's non-volatile memory. The option ROM may communicate its determination to BIOS, as illustrated in block 350.

In block 360, the executed BIOS code invokes option ROM code to filter disk requests when a disk request is made for a disk drive that is cached. The executed option ROM 112 code may at this point monitor the system for write requests, as in block 215 in Fig. 2 and in block 360 in Fig. 3, and also log write requests to non-volatile memory, as in block 225 of Fig. 2. The executed option ROM 112 code may return to BIOS data which tells BIOS, as shown in block 370, to execute or fail the write request, or that the option ROM will service the request.

For logged write requests, the option ROM may record the location on the disk, but does not save the actual disk data. Then the option ROM may cause the normal BIOS interrupt 13 disk write routine to perform the requested disk write.

In another embodiment, an option ROM 112 (Fig. 1) may monitor the write requests of block 215 in Fig. 2 by having the executed option ROM code modify the processor stack. The executed option ROM 112 code may modify the processor stack, in one example, by using the algorithm 400 in Fig. 4. In block 410, the executed option ROM code initializes as a normal interrupt 13 handler by identifying disk drives that it supports. After the BIOS code finishes its initialization of the option ROM and allows the option ROM code to make interrupt 13 disks requests for the disk drives that are supported by the option ROM, the executed option ROM code then determines the stack offset, as illustrated in block 420. To determine the stack offset, in one example, the executed option ROM code stores the current stack location and then invokes a disk request for a disk drive that is supported by the subject option ROM. The option ROM code, since it is in the initialization process, saves the stack pointer at the point of invocation and returns with failure response. Upon return, a stack finder code can determine how far down the stack the return instruction pointer and return code segment are from the option ROM's interrupt 13 invocation. This may be the stack offset. The option ROM code may then set a flag so that future invocations of the option ROM can be normally processed.

During subsequent interrupt 13 write requests such as in Block 430, the executed option ROM code may replace the instruction pointer and code segment at the determined stack offset with another option ROM execution address, as illustrated in block 440. In block 450, the executed option ROM code returns control to the BIOS code which will in turn return to what it thinks is the original requester of an interrupt 13 disk request operation. However, since the stack instruction pointer and code segment have been changed, control will actually revert back to the option ROM code, as illustrated in block 460. Therefore by modifying the stack, the option ROM may, as illustrated in block 470, monitor write requests (as in block 215, Fig. 2), log write requests (block 225, Fig. 2), or execute write to disk requests (block 230, Fig. 2).

While the present invention has been described with respect to a limited number of embodiments, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of this present invention.

What is claimed is: